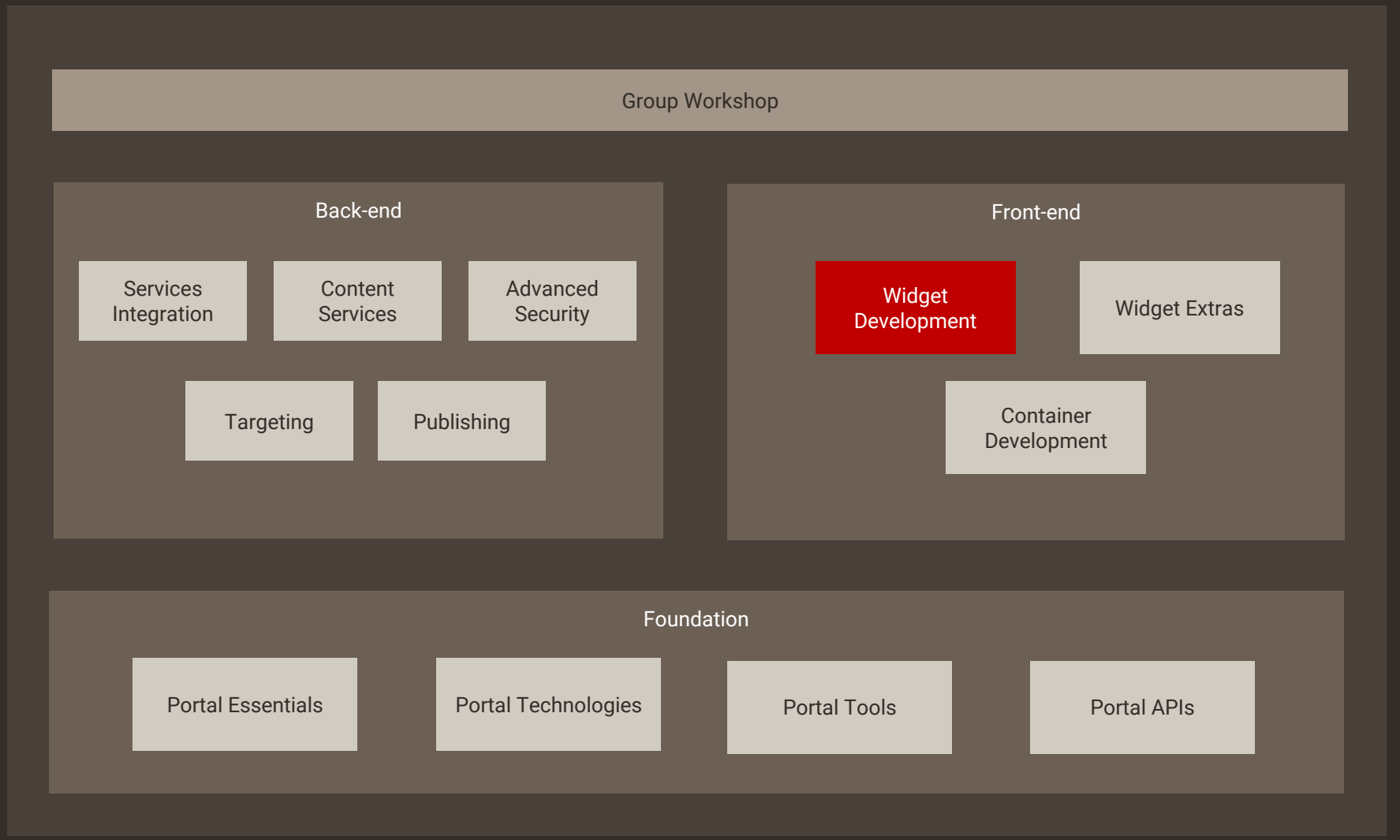
BACKBASE

Widget Development

Focus Area: Front-end

Training Overview

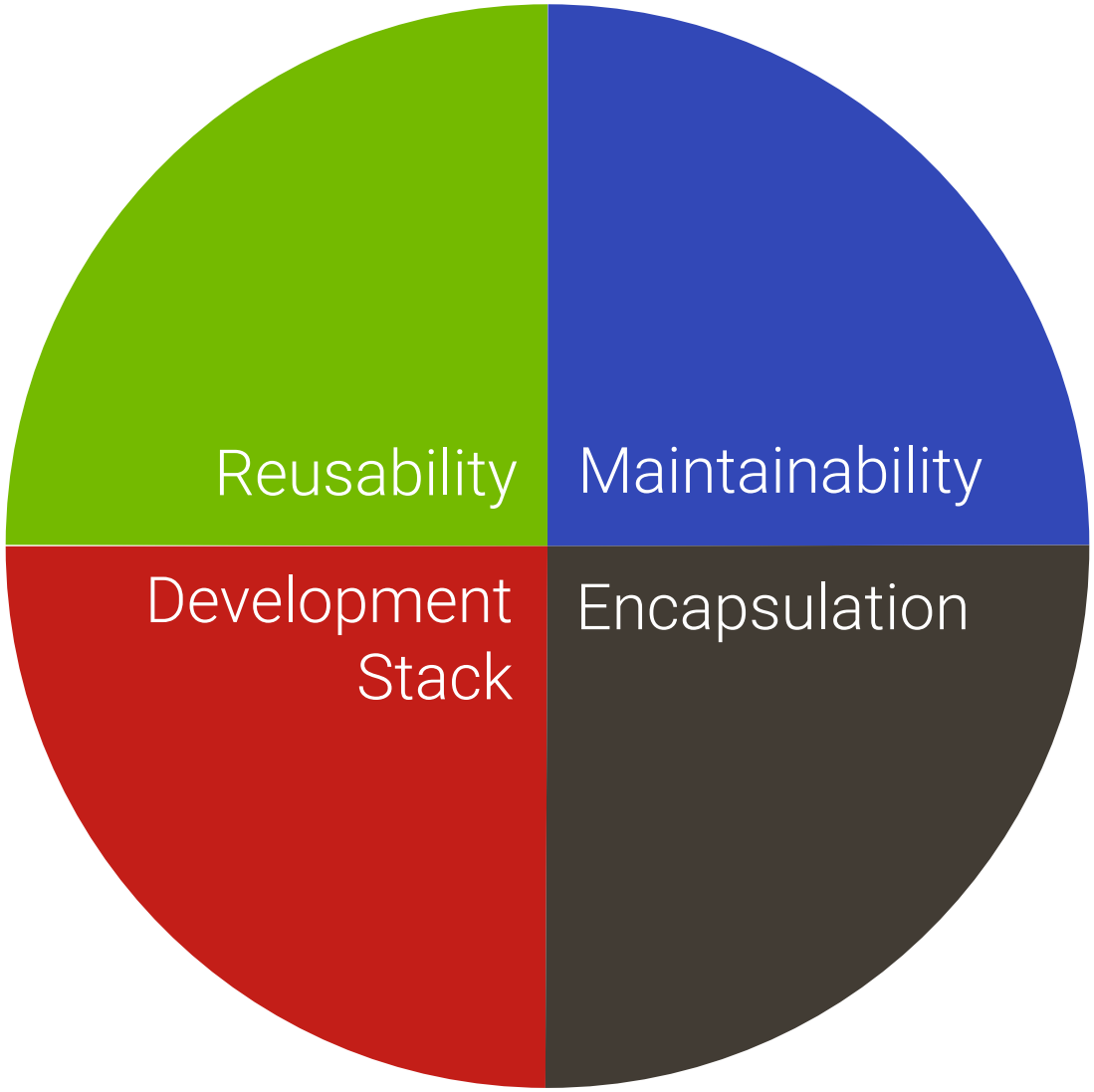


1. Understanding Widgets
2. Working with Widget Collection 2
3. Extending Widget Functionality
4. Theming your Portal
5. Developing Custom Widgets

BACKBASE

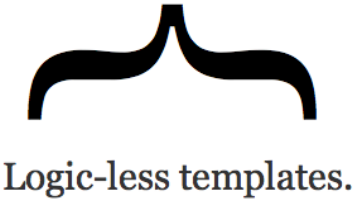
Understanding Widgets

*"Widgets are **autonomous** mini-**applications**, designed to run within a web page. They implement **reusable** pieces of **business logic**, with which you can compose your web application."*


Reusability
Maintainability
Development Stack
Encapsulation


HTML


XML


Logic-less templates.


CSS


B
Bootstrap


Sass


JS


A


jQuery
write less, do more.


bb-cli

bb-cli

bb-cli is a command line tool provided and supported by Backbase to ease front-end development for Backbase Platform.

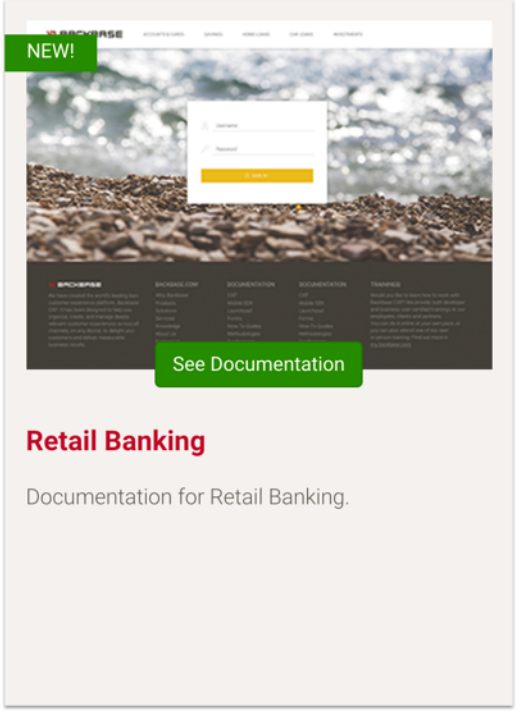
- Scaffolding
- Importing
- Packaging
- Unit Test
- and more...

BACKBASE

Widget Collection 2

Widget Collection Retail Banking 2

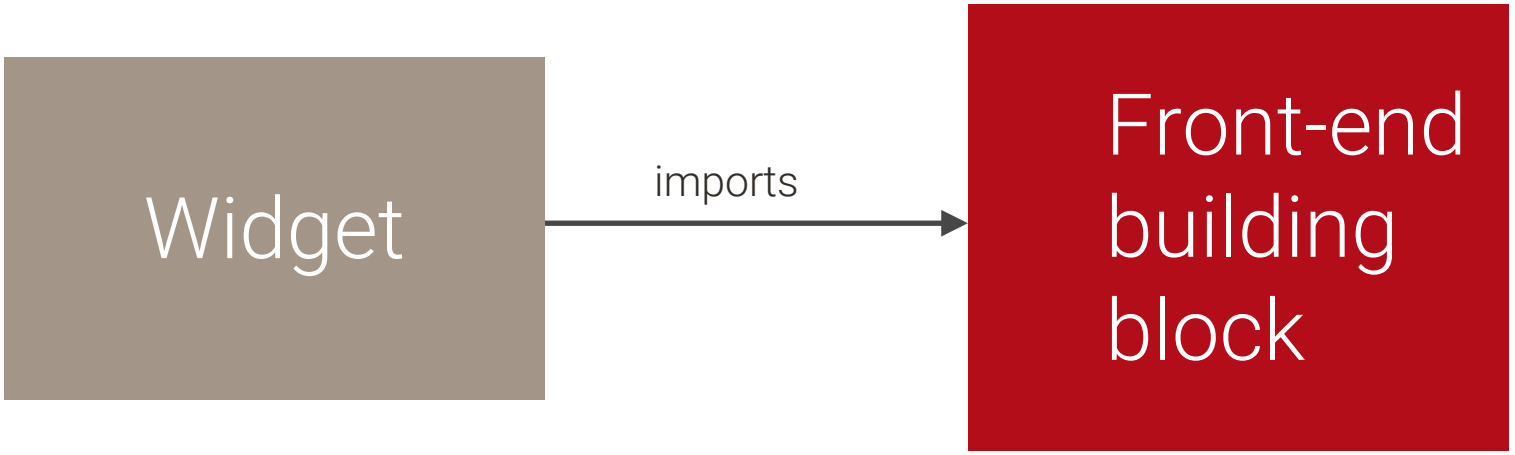
- Out-of-the-box Functionality
  - Login, Transactions List, Product Summary and many more...
  - Covers Essential Use Cases
- Extendable & Upgradable
- Frontend Building Blocks



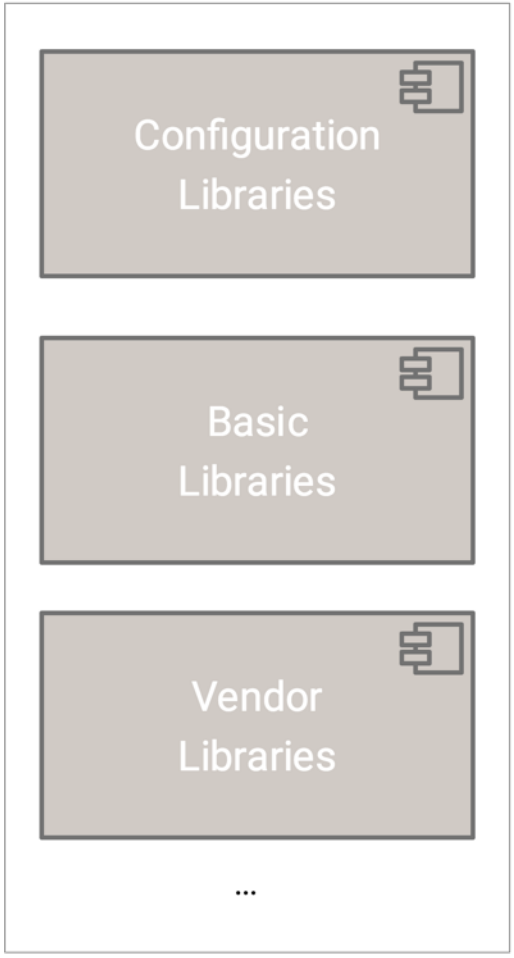

RETAIL BANKING
PORTAL DEMO

Frontend Building Blocks

*"The building blocks are the **infrastructure** that enables the implementation of the **architecture**."*



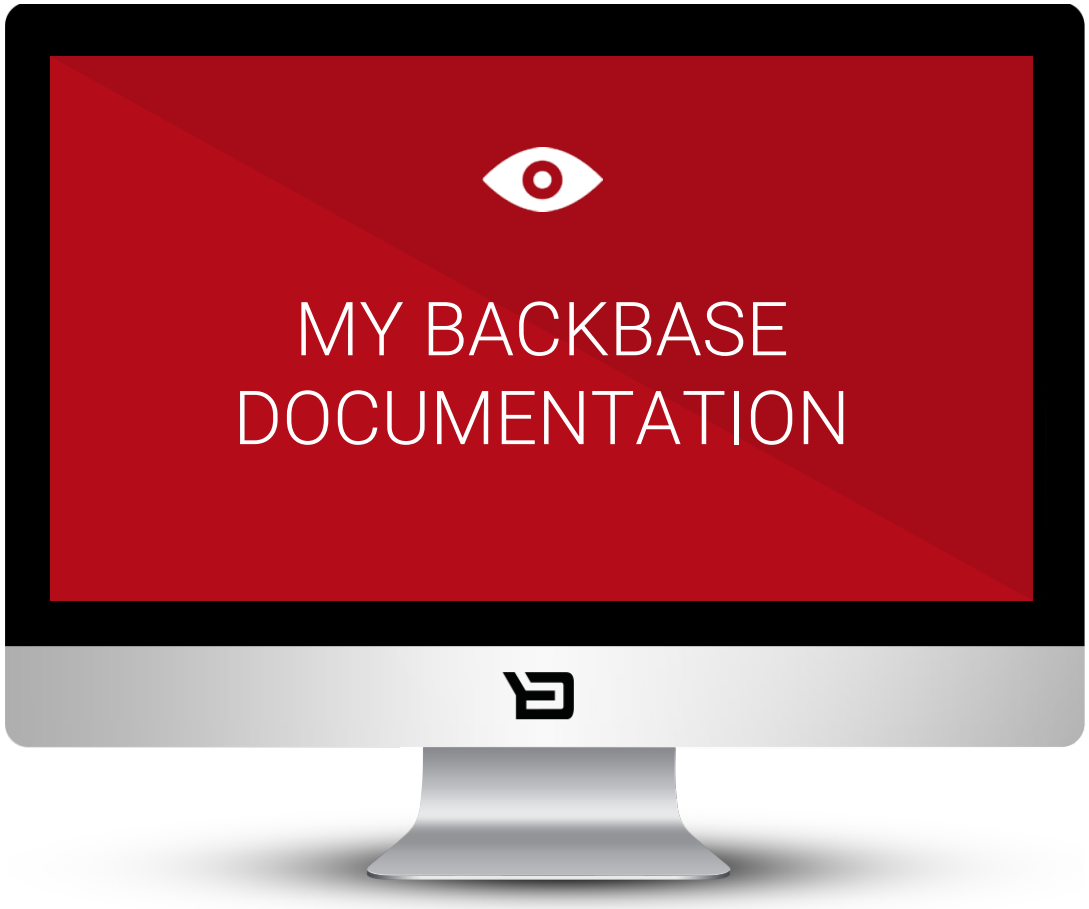
Frontend Building Blocks



- Configure i18n
- Configure Base URI for Data-Modules

- Event Bus
- Storage
- Error Handling

- AngularJS
- UI Bootstrap
- SystemJS


MY BACKBASE
DOCUMENTATION

What is the naming convention?

{type}-{project}-{name}-{subname}*-{meta}

widget-bb-product-summary-ng

Setup your Development Environment

*Before developing your application, we need to install the **tools** that we use in our workflows and setup the **training portal**.*

1. Install and configure additional bb-cli commands.

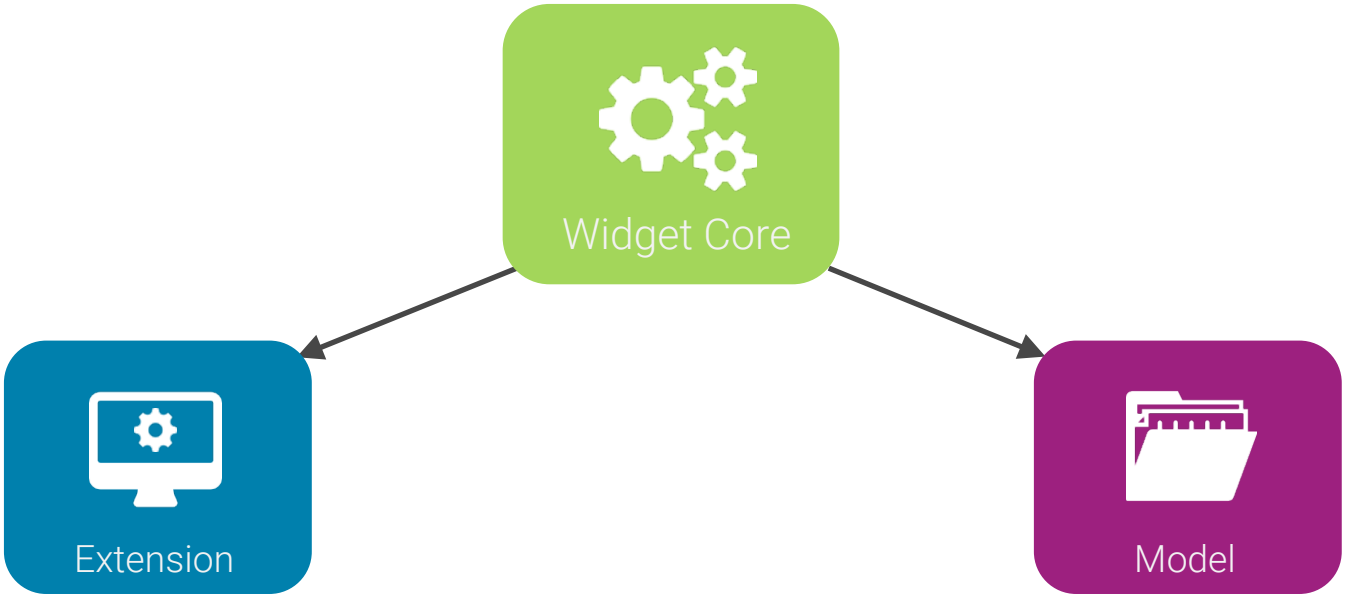
2. Setup the Training Portal.

BACKBASE

Widget Architecture

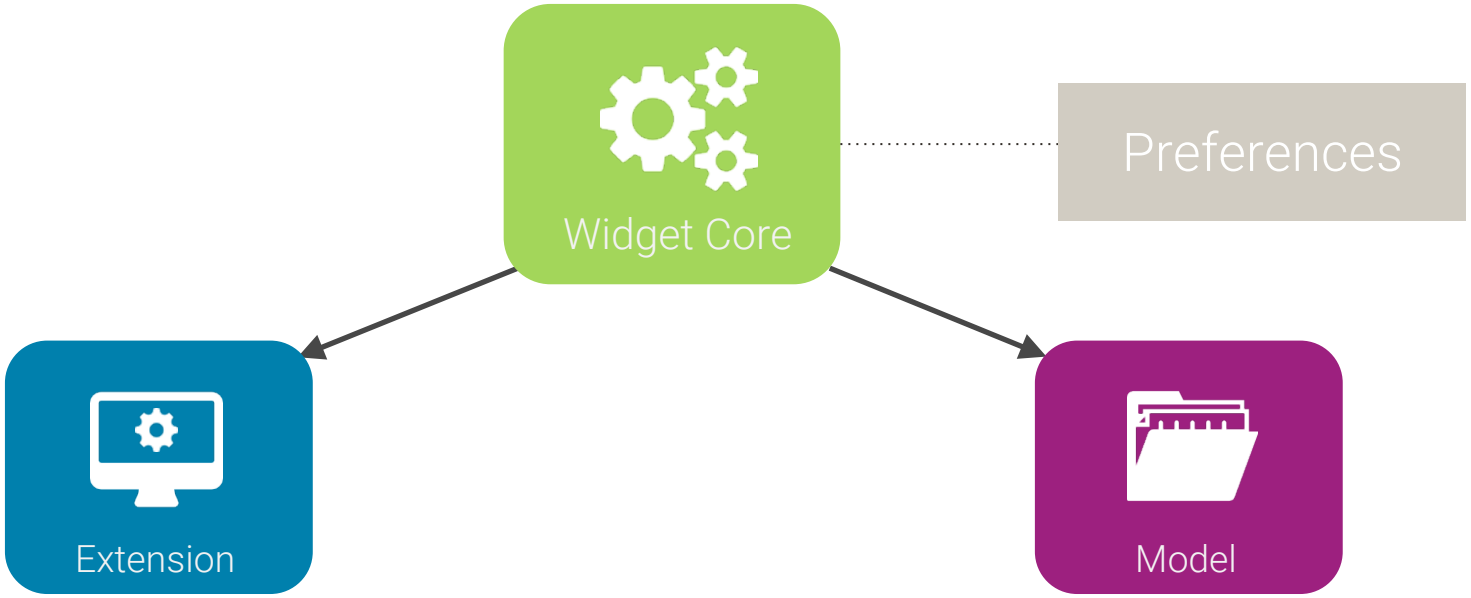
Widget Collection 2

Widget Architectural Overview




HELLO WORLD WIDGET

Widget Architectural Overview



Adding a Custom Property to Model XML

```
<catalog>
        <widget>
                <name>widget-training-hello-world-ng</name>
                <contextItemName>[BBHOST]</contextItemName>
                <properties>
                        ...
                <property name="defaultView" label="Default View">
                        <value type="string">list</value>
                </property>
                         ...
                </properties>
        </widget>
</catalog>
```

Configure preferences with *viewHint*

```
<property name="defaultView" label="Default View" viewHint="designModeOnly,admin,text-input" >
          <value type="string">list</value>
</property>
```

String composed of the following optional, comma-separated parts

- [designmode]: *Where the property can only be edited within CXP Manager*
- [role]: *The role needed to edit the property via the preference form*
- [input-field]: *What input field is displayed in the preference form*


CUSTOM PREFERENCES

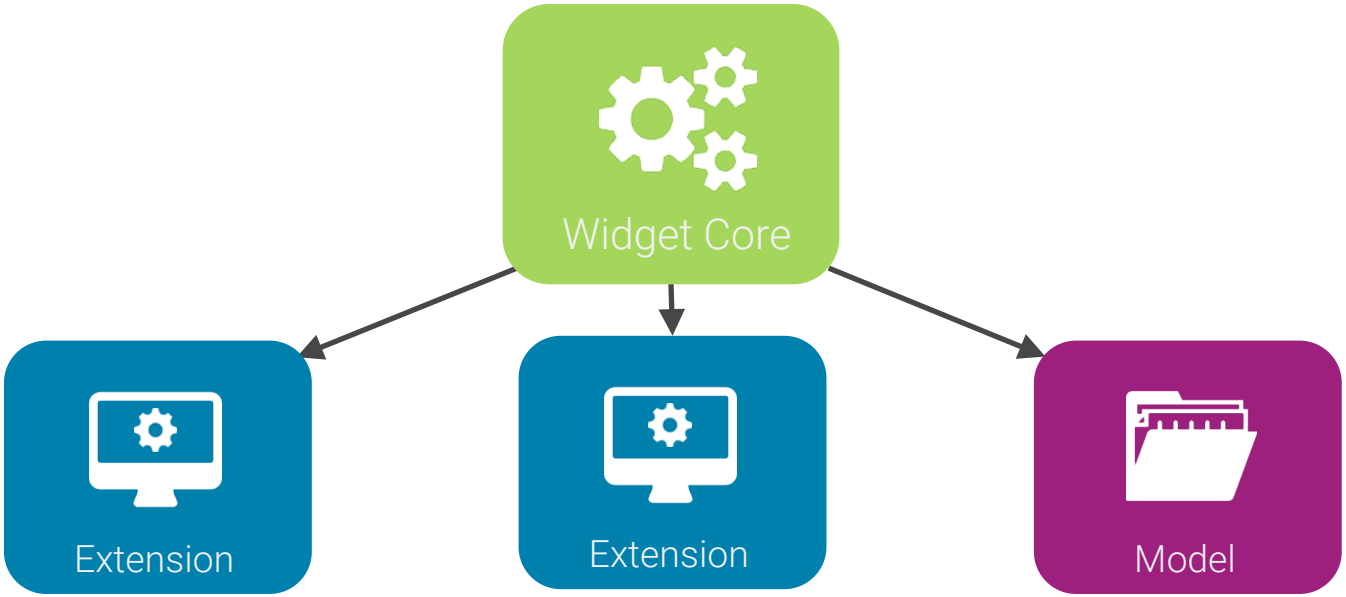
BACKBASE

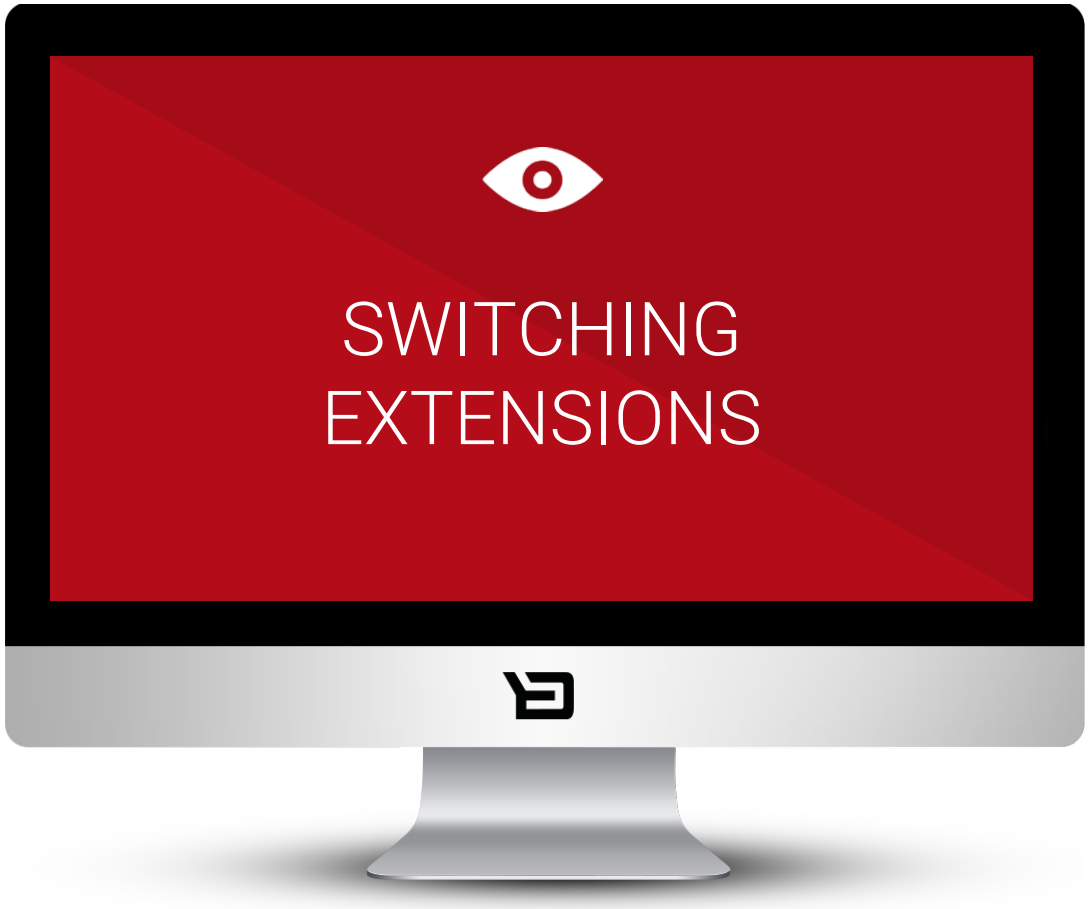
Understanding Extensions

Extensions Overview

*"An extension allows developers to **extend** a widget's **logic** and **customize** its **presentation**."*

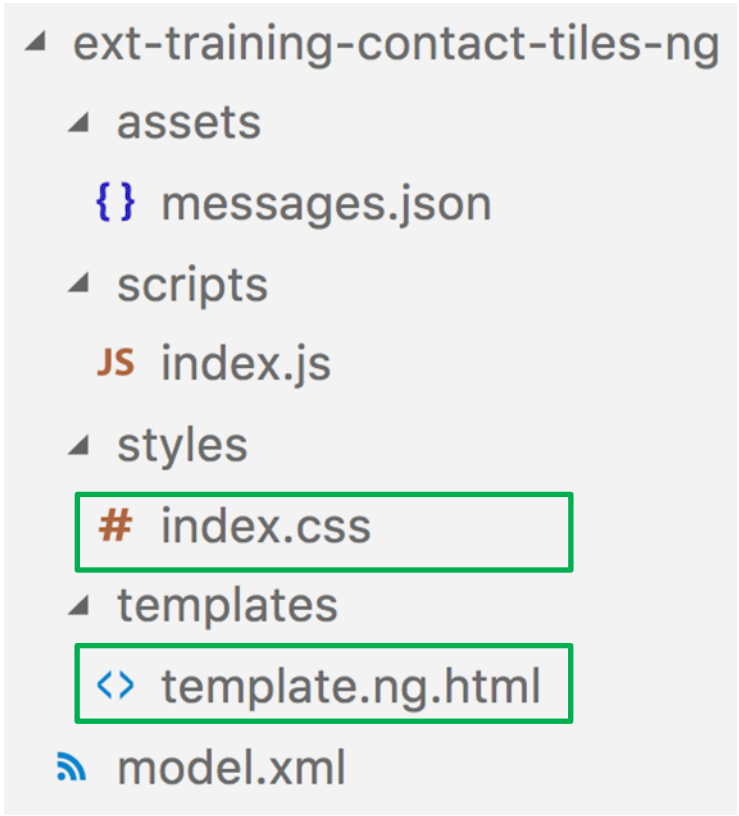
Widget Architectural Overview




SWITCHING
EXTENSIONS


GENERATE AN
EXTENSION

Generated Extension



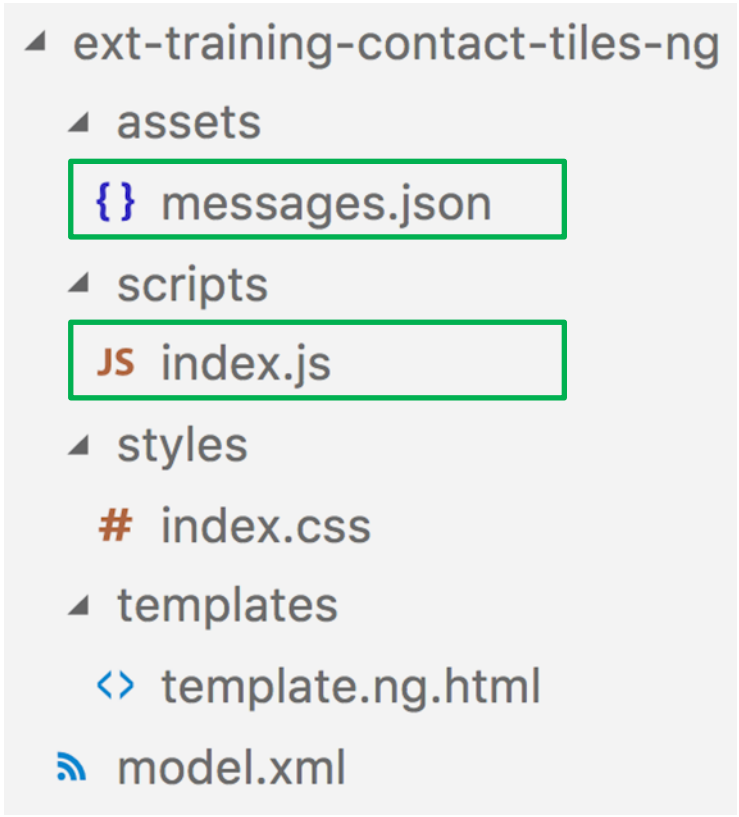
Styling

CSS

Presentation

AngularJS Template

Generated Extension



i18n

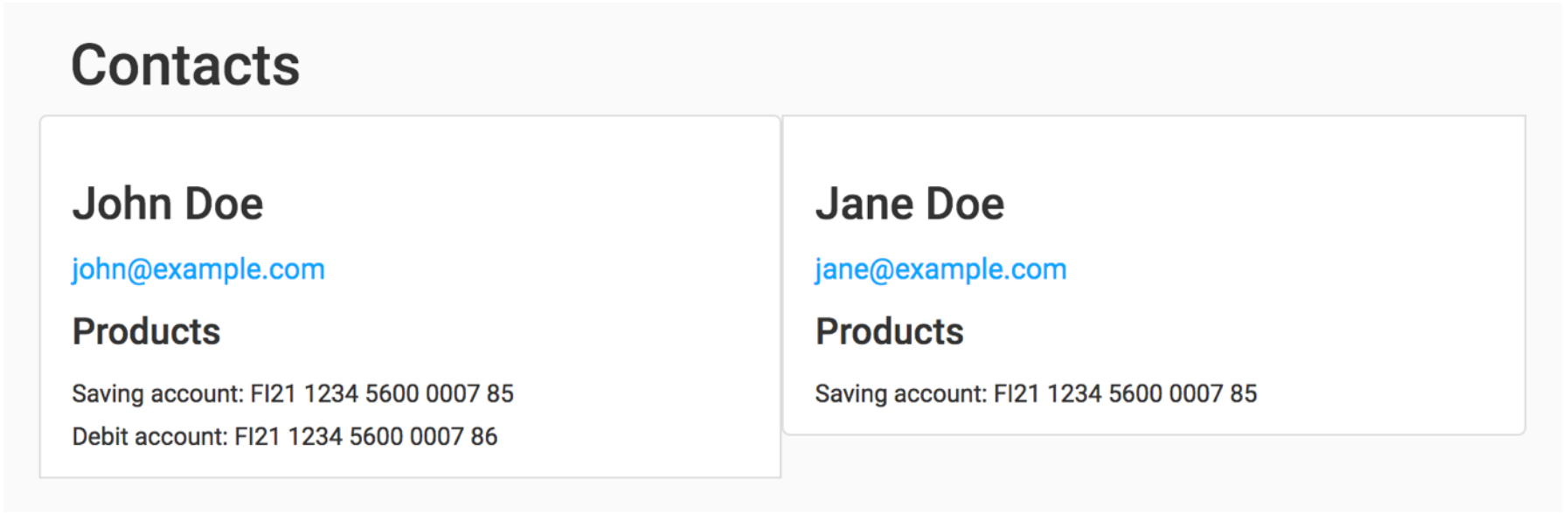
JSON

Behavior

Inject Dependencies
Implement Hooks
Implement Helpers
Handle Events

Create an Extension

*Currently the **Contact Manager Widget** has an extension that shows a **list** view. Create a new extension to display the contacts as **tiles**.*



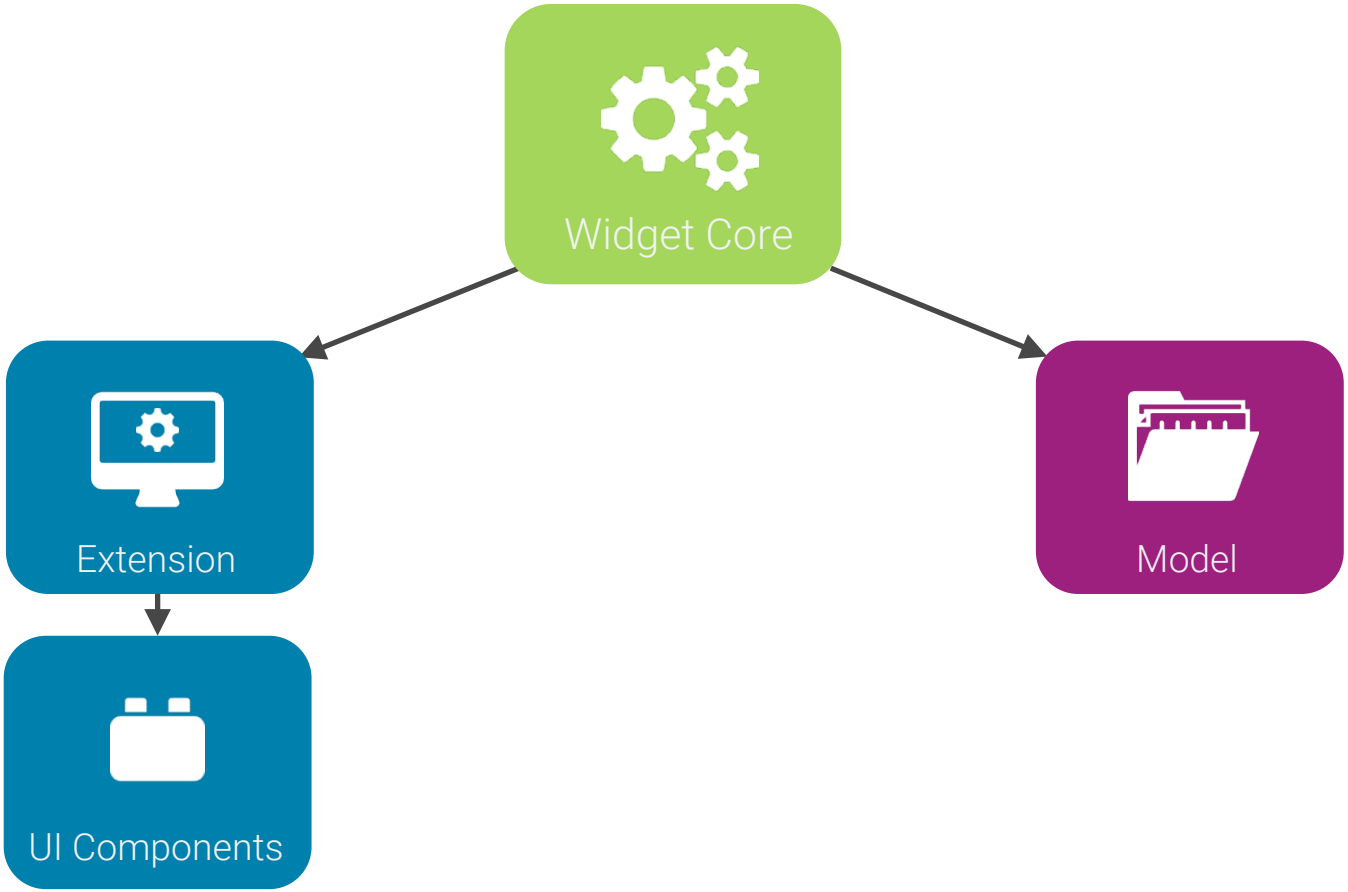
BACKBASE

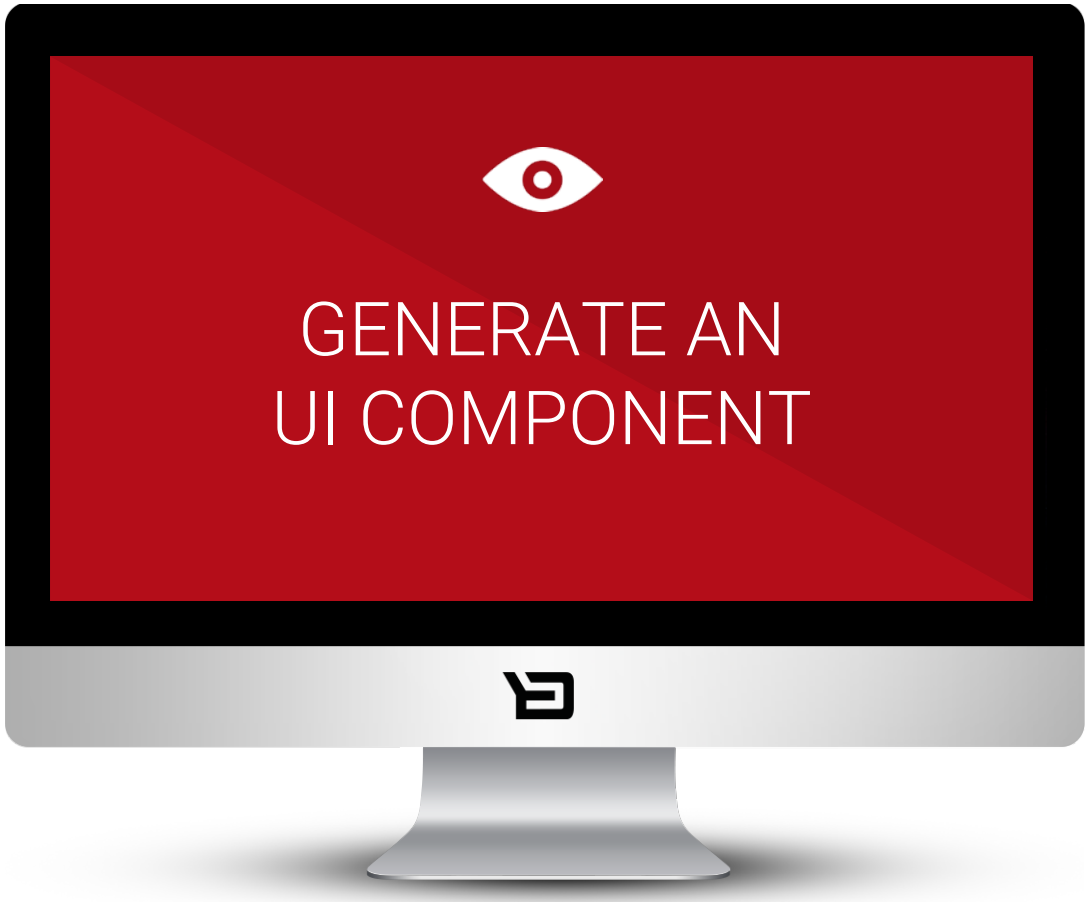
UI Components

Extending Widget Functionality

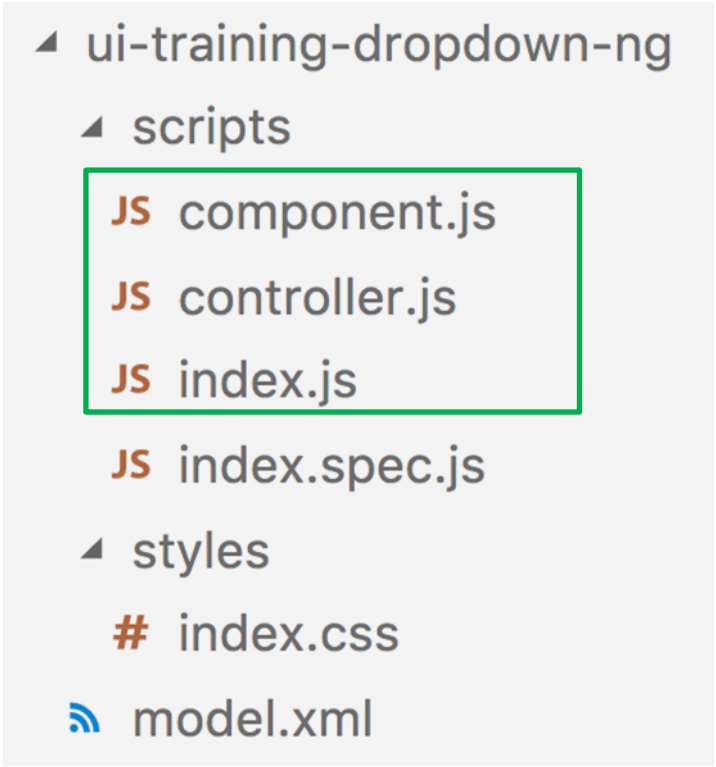
*”UI Components **maximize** the **reusability** of frontend code, allowing the developer to reuse UI and their logic in **different widgets**”*

Widget Architectural Overview




GENERATE AN
UI COMPONENT

Generated UI Component



Component

Bindings, Template, etc.

Controller

Component Logic

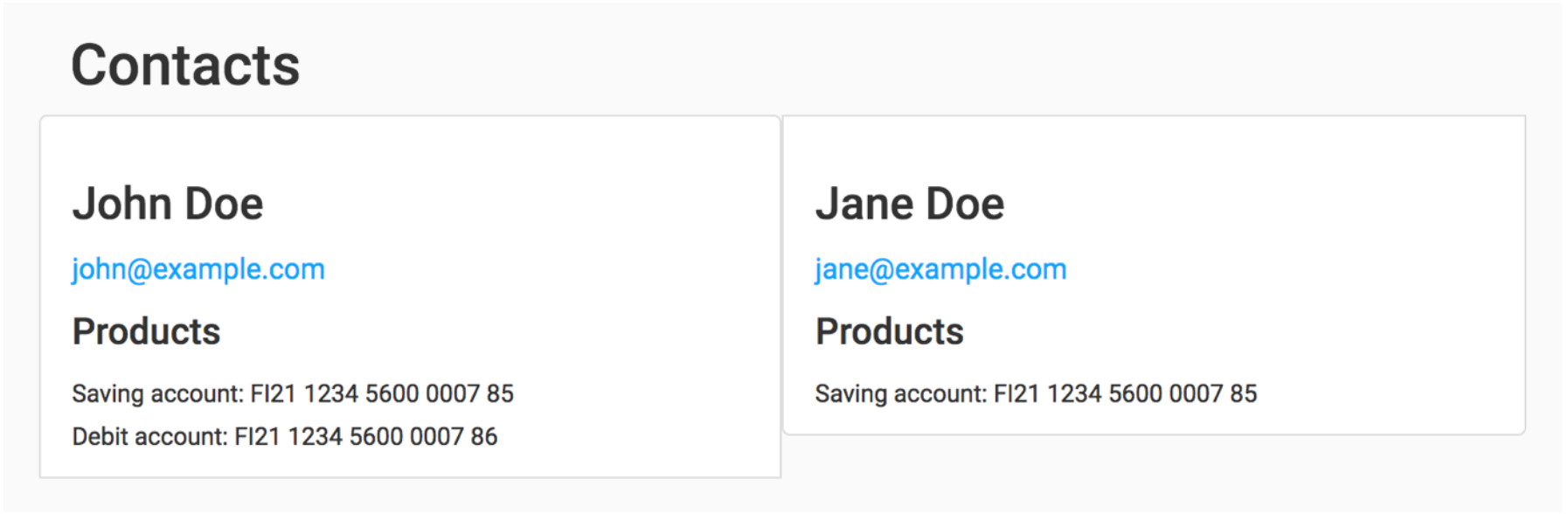
Index

Register Component

Inject Dependencies

Create an UI Component

*Using the previous exercise, create an **UI Component** that displays all the contacts information (Name, IBAN, etc.). Then, update your extension template in order to use it.*



BACKBASE

Backbase Themes

Widget Development

Backbase Themes

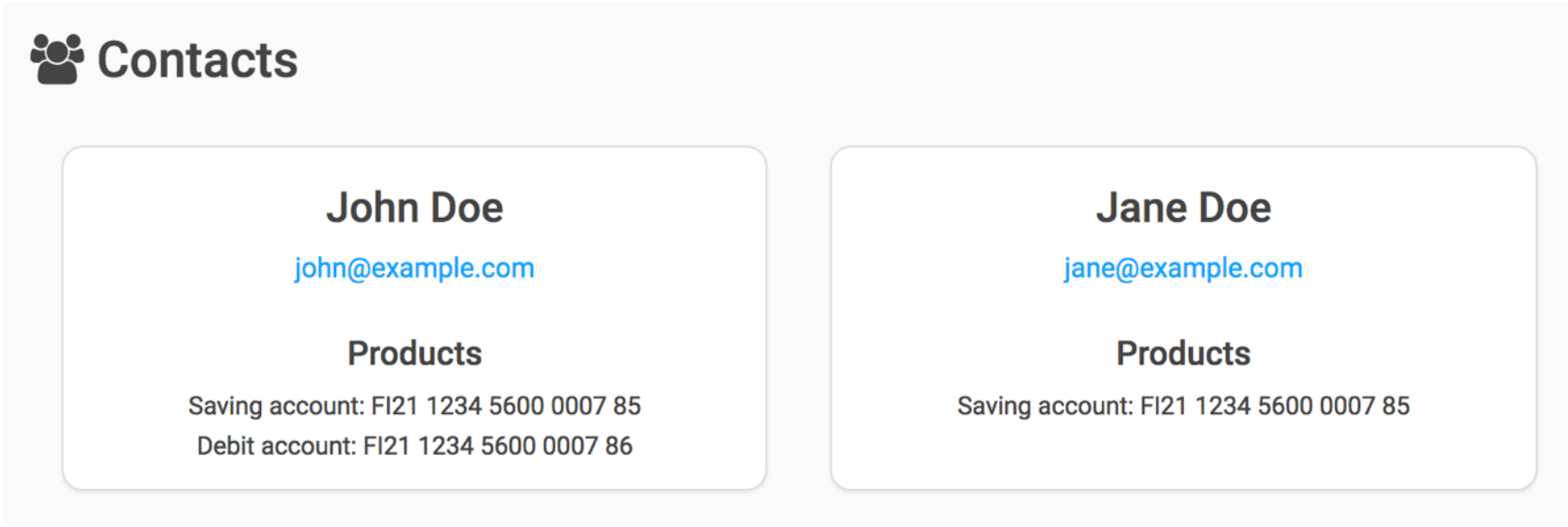
VS

Widget Styles


CUSTOMIZING
RETAIL THEME

Theming your Portal

The main propose of this exercise is, using a new theme for your portal, to add style into your components.



BACKBASE

Extending Widget Functionality

Understanding Extensions

Objectives

1. Modifying data from Widget Core
2. Adding extra logic in the Extension

BACKBASE

Hooks

Extending Widget Functionality

*“Hooks are **defined places** within a **widget-core** from which a developer's code can be called. This allows the extension to **change the behavior** of the widget-core in someway.”*

When to use an Extension Hook

- **Transform** received data or data to be sent
- **Manipulating behaviors** of the Widget Core on certain events

Keep in mind when using Extension Hook

- Only **available** if it is implemented in the Widget Core
- Every hook provides a **specific set of data** to the implementation
- The available hooks list can be found in the **Widget's Documentation**

Contact Details Widget Hooks

Name	Trigger	Data
deleteContact	On user initiating delete action.	Contact.
getSelectedContact	On loading contacts.	List of contacts and contact to be selected.
selectPrevContact	Preparation for deleting a contact.	List and contacts and index of selected contact.
processContacts	On returning contacts.	List of contacts.
processSearchContacts	Search completed.	List of contacts returned by search.
…	…	…

Context Object

- **widget**: The widget object.

```
const NAME_PREFIX_PREFERENCE = 'namePrefix';
export const hooks = ({widget}) => {
    const NAME_PREFIX = widget.getStringPreference(NAME_PREFIX_PREFERENCE);
    return {
        processContacts: (contacts) => contacts.map(contact =>
            Object.assign({}, contact, { name: `${NAME_PREFIX}${contact.name}` })
        )
    };
});
```

Contact Details Widget Hooks

#processContacts(contacts)

Purpose:
Intended to allow the Developer to process list of contacts.

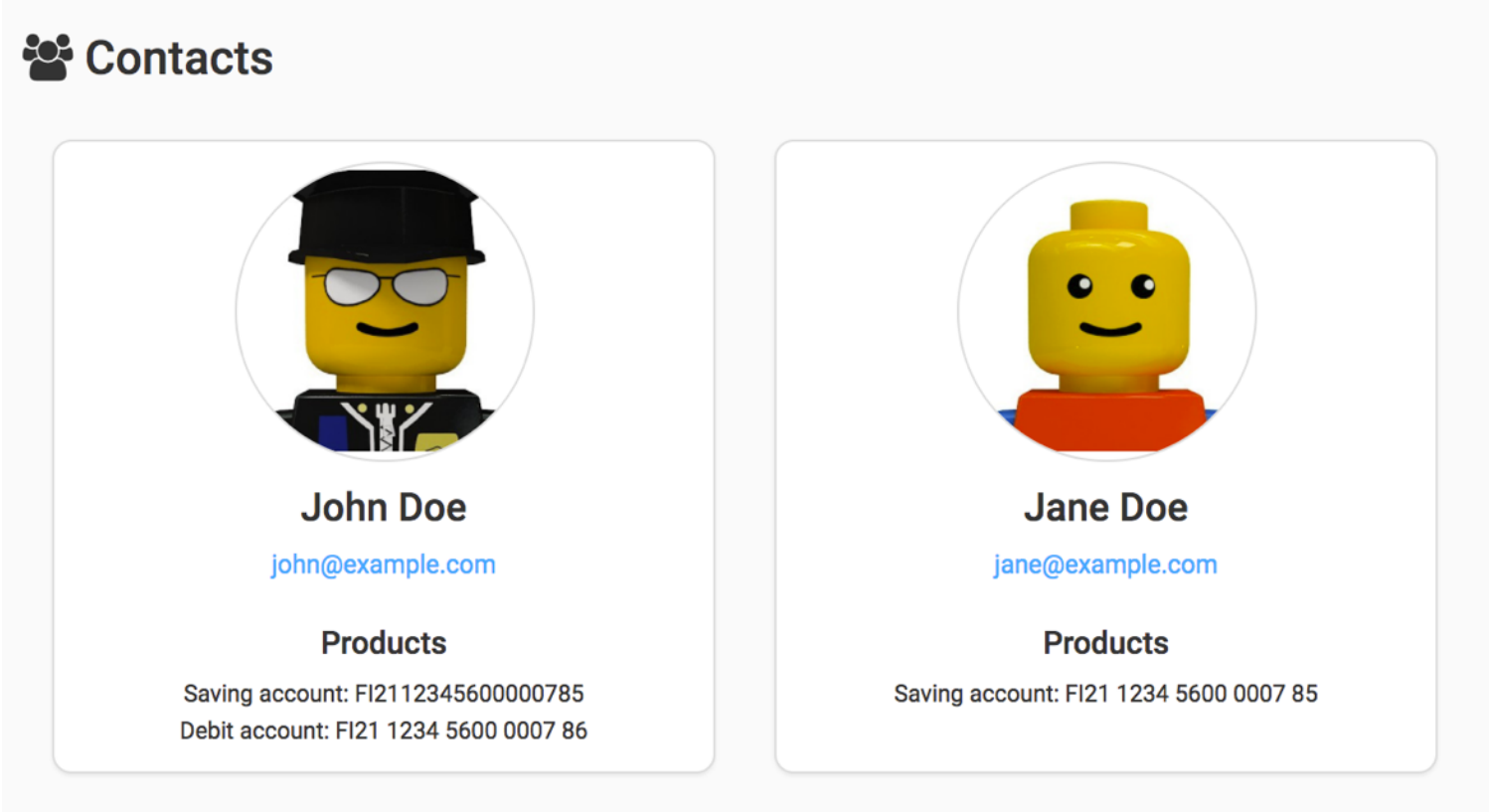
Return:
Return the array of updated contacts.

```
const NAME_PREFIX_PREFERENCE = 'namePrefix';
export const hooks = ({widget}) => {
    const NAME_PREFIX = widget.getStringPreference(NAME_PREFIX_PREFERENCE);
    return {
        processContacts: (contacts) => contacts.map(contact =>
            Object.assign({}, contact, { name: `${NAME_PREFIX}${contact.name}` })
        )
    };
});
```


HOOKS

Implement an Extension Hook

The objective is to add the avatar image for each contact. We will need to get this from an external source. Using Hooks we will be able to manipulate the widget behavior.



BACKBASE

Helpers

Extending Widget Functionality

*“Sometimes you need some **extra logic** in your view that is **not part** of the **controller interface**. For this you can use **helpers** to move view logic out of the template.”*

Context Object

- $filter: The angular $filter object
- publish: The publish function. (Pub / Sub)
- widget: The widget object.

```
export const helpers = ({$filter, publish, widget}) => ({
  isLowBalance:(product) => product.availableBalance <
widget.getLongPreference('lowBalanceThreshold')
});
```

Implement an Extension Helper

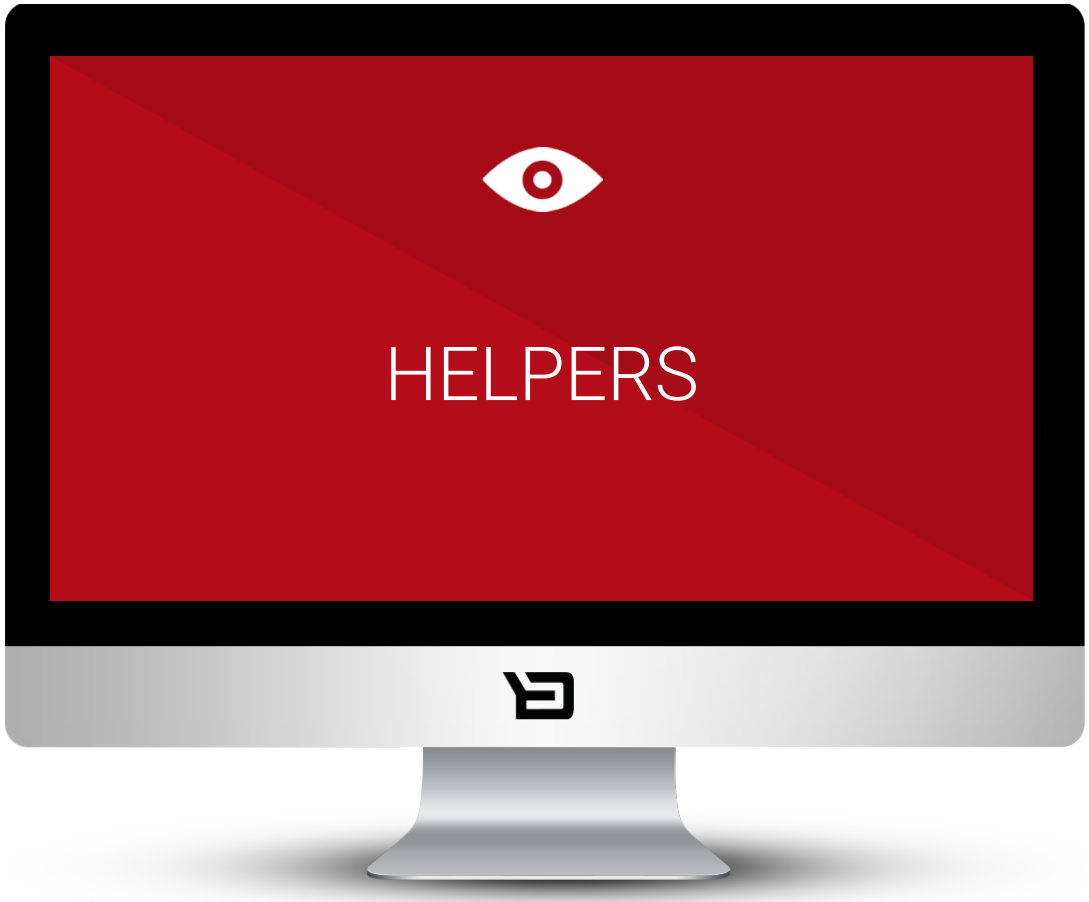
Example:

Highlighting products that have balance below zero.

```
export const helpers = ({$filter, publish, widget}) => ({
  isLowBalance:(product) => product.availableBalance <
        widget.getLongPreference('lowBalanceThreshold')
});
```

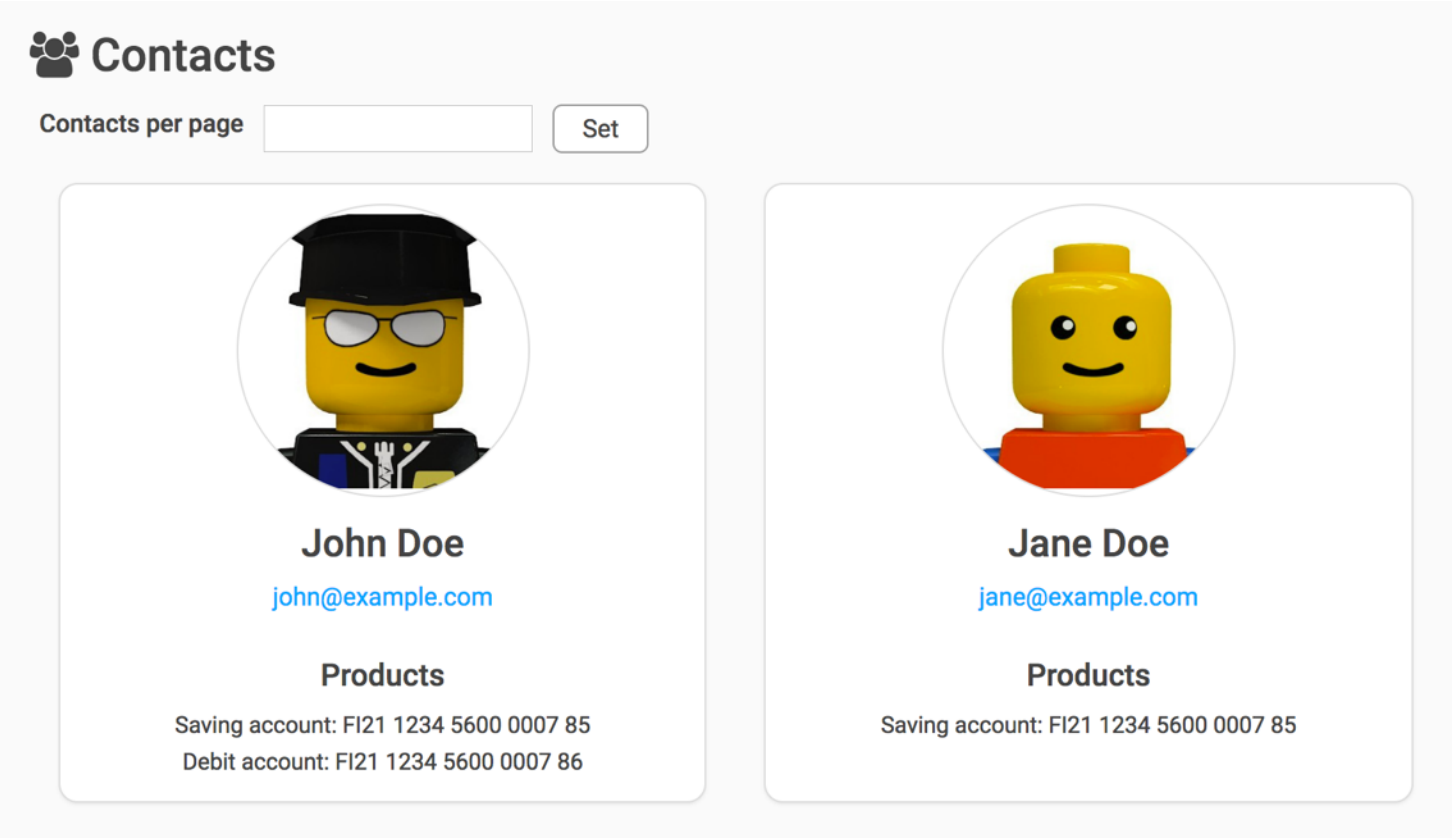
The helpers are made available on the root scope on **ext.helpers.<helper-name>**

```
<span ng-class="{low-balance: ext.helpers.isLowBalance(product)}"></span>
```


HELPERS

Implement an Extension Helper

*The objective of this exercise is to, using the **Contact Manager Widget**'s preference **pageSize**, make the property editable in the widget extension template.*

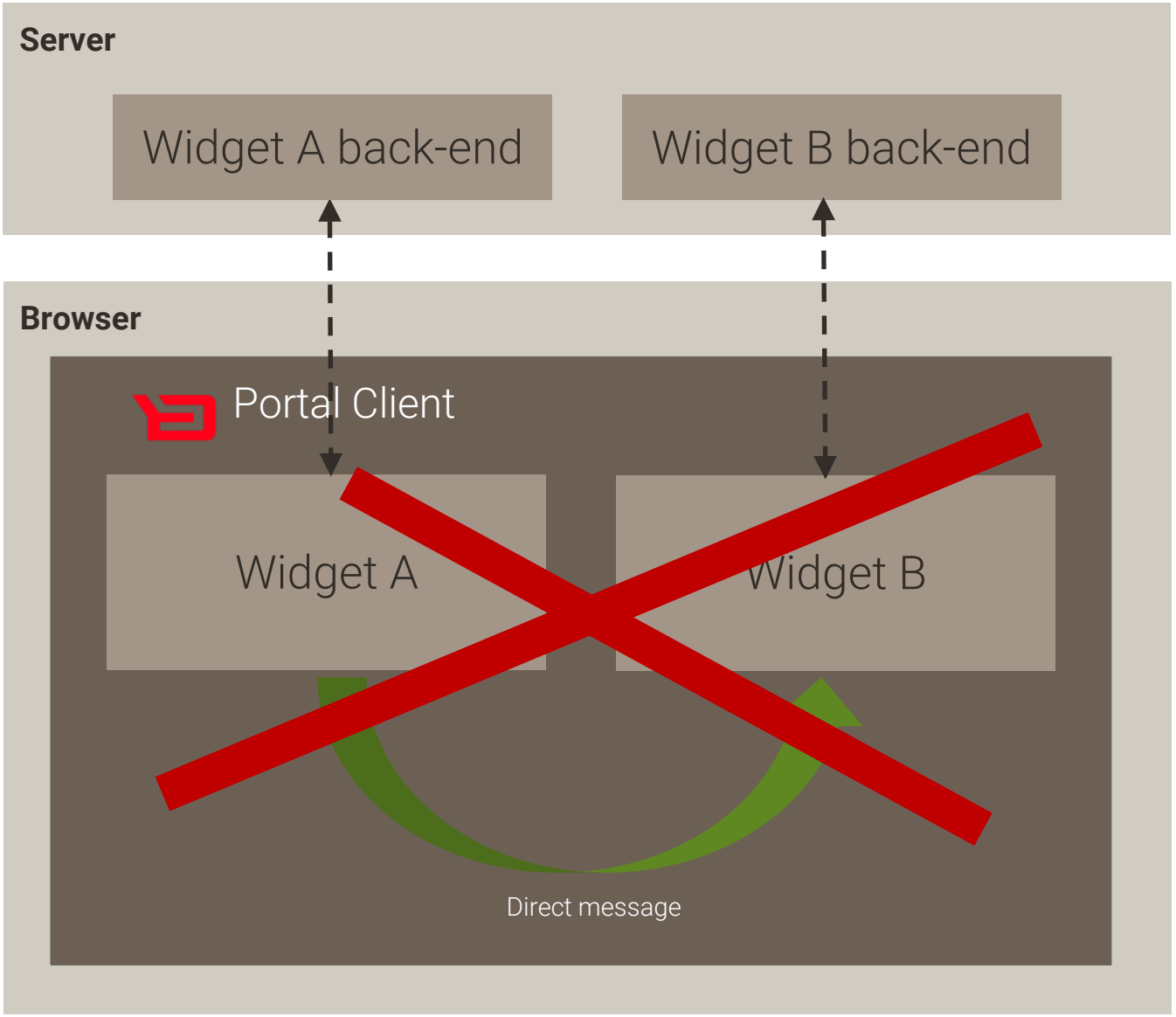


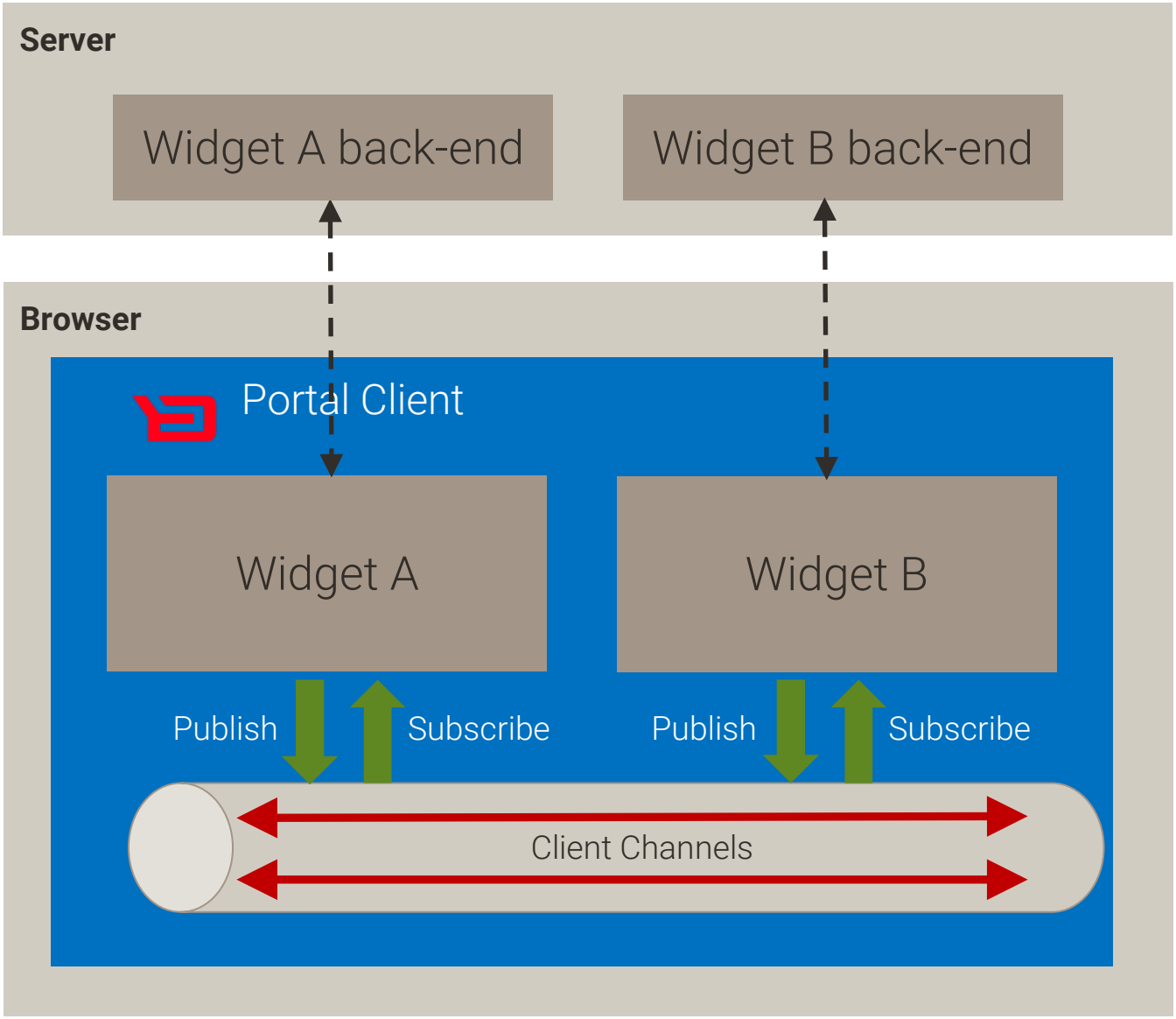
BACKBASE

Events

Extending Widget Functionality

*“The **Publish** and **Subscribe** pattern, often referred to simply as PubSub is used to **pass information between widgets**.”*


Server
Widget A back-end
Widget B back-end
Browser
Portal Client
Widget A
Widget B
Direct message


Server
Widget A back-end
Widget B back-end
Browser
Portal Client
Widget A
Widget B
Publish
Subscribe
Publish
Subscribe
Client Channels

Contact Manager Widget Events

Event	Action	Event Message
bb.event.contact.create.done	Contact creation finished	None
bb.event.contact.create.failed	Contact creation failed	Error object
bb.event.contact.delete.done	Contact deletion finished	None
bb.event.contact.delete.failed	Contact deletion failed	Error object
...	...	...

Context Object

- $filter: The angular $filter object
- publish: The publish function. (Pub / Sub)
- widget: The widget object.

```
export const events = ({ $filter, publish, widget }) => {
    "bb.event.product.selected": (product) => window.location.href =
`/products/${product.id}`
};
```

Subscribing to a Widget Event

bb.event.product.selected

Purpose:
Published when a user has selected a product.

Arguments:
Selected product.

```
export const events = ({ $filter, publish, widget }) => {
    "bb.event.product.selected": (product) => window.location.href =
`/products/${product.id}`
};
```

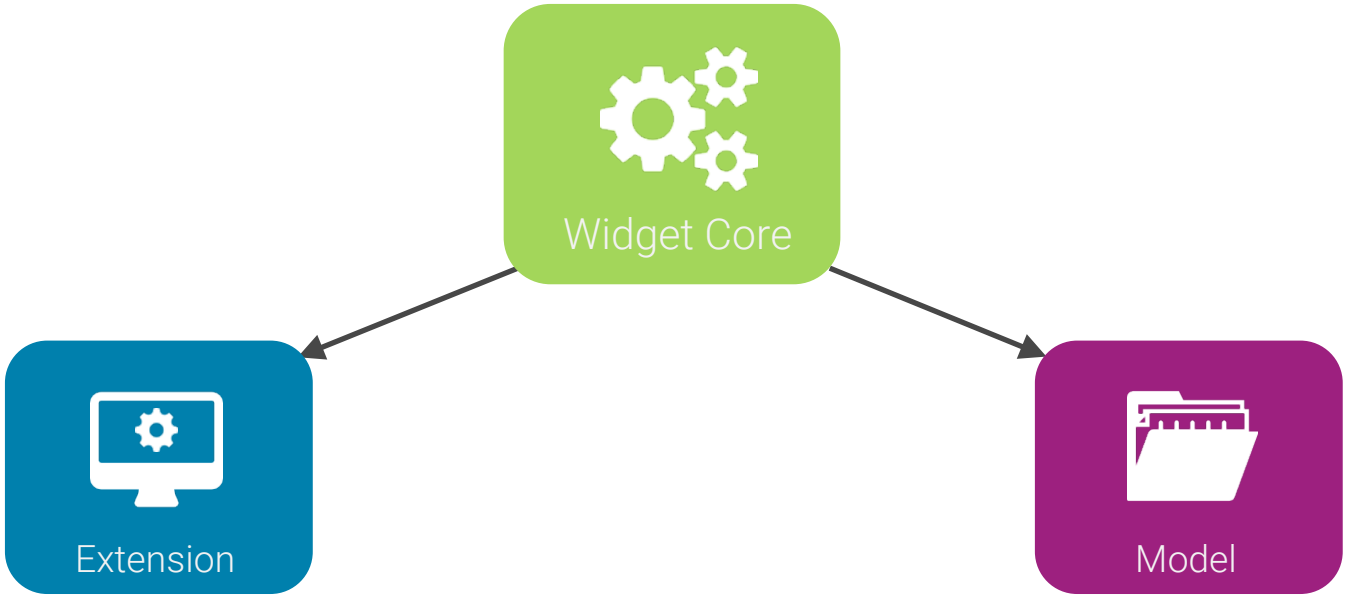
BACKBASE

Custom Widget

Generating a new Widget

1. Understanding Widget Core, Widget Model & Data Modules
2. Developing Custom Widget

Widget Architectural Overview



Widget Development

Create a new widget for Appointments that displays the list of appointments scheduled by customers with their advisors.

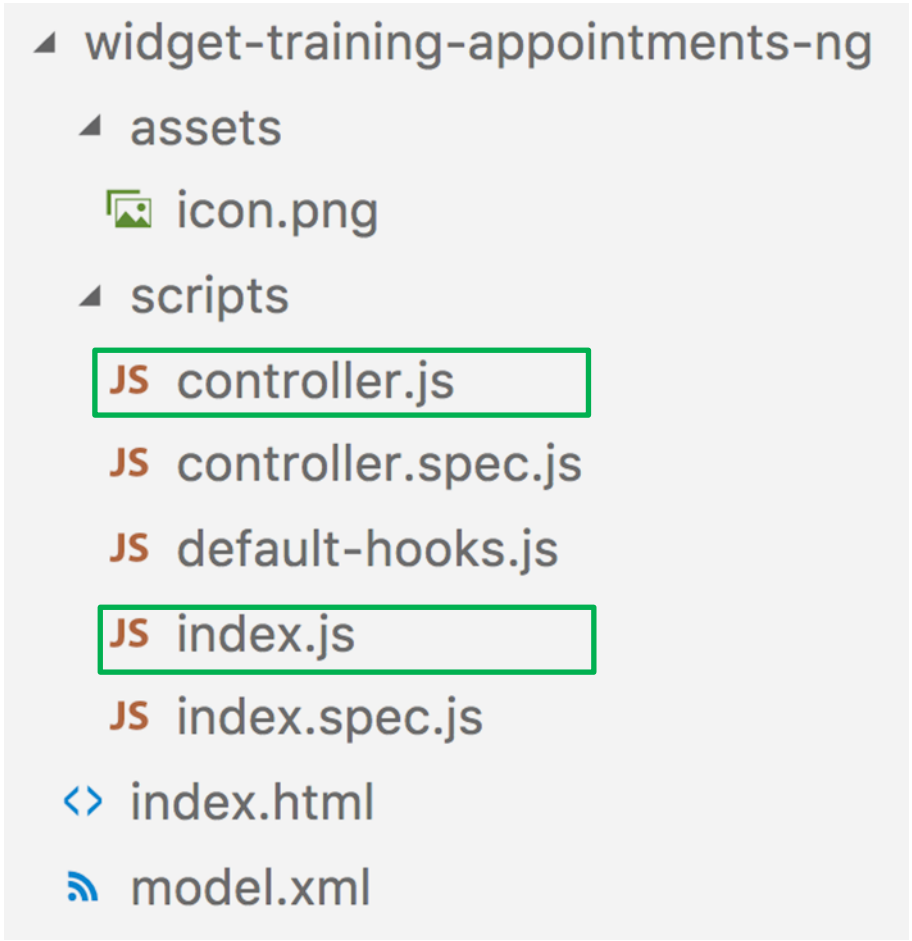
Appointments List			
31/01/2019	14:26	Cline Glass Savannah Briggs Franks Frank	Meeting with Mr.Glass
17/03/2017	07:43	Shawn Cooley Crystal Mccoy	Urgent - Meeting with Ms.Cooley
19/11/2017	13:50	Miriam Sims Drake Whitley Whitfield Ballard	Meeting with Ms.Sims
23/12/2016	23:42	Concepcion Underwood	Meeting with Ms.Underwood
25/07/2020	11:05	Wynn Battle Espinoza Fleming	Urgent - Meeting with Mr.Battle

BACKBASE

Widget Core

Widget Development

Scaffolded Widget Core



Controller

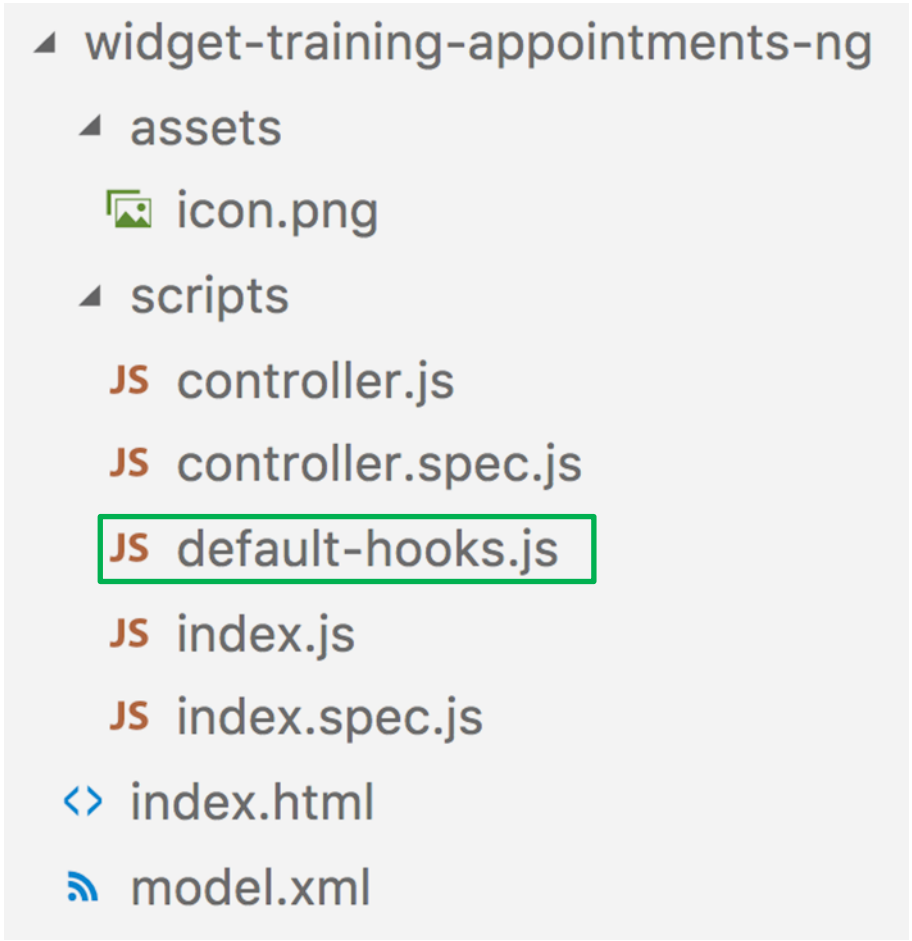
Business Logic

Index

Register as a Module

Inject Dependencies

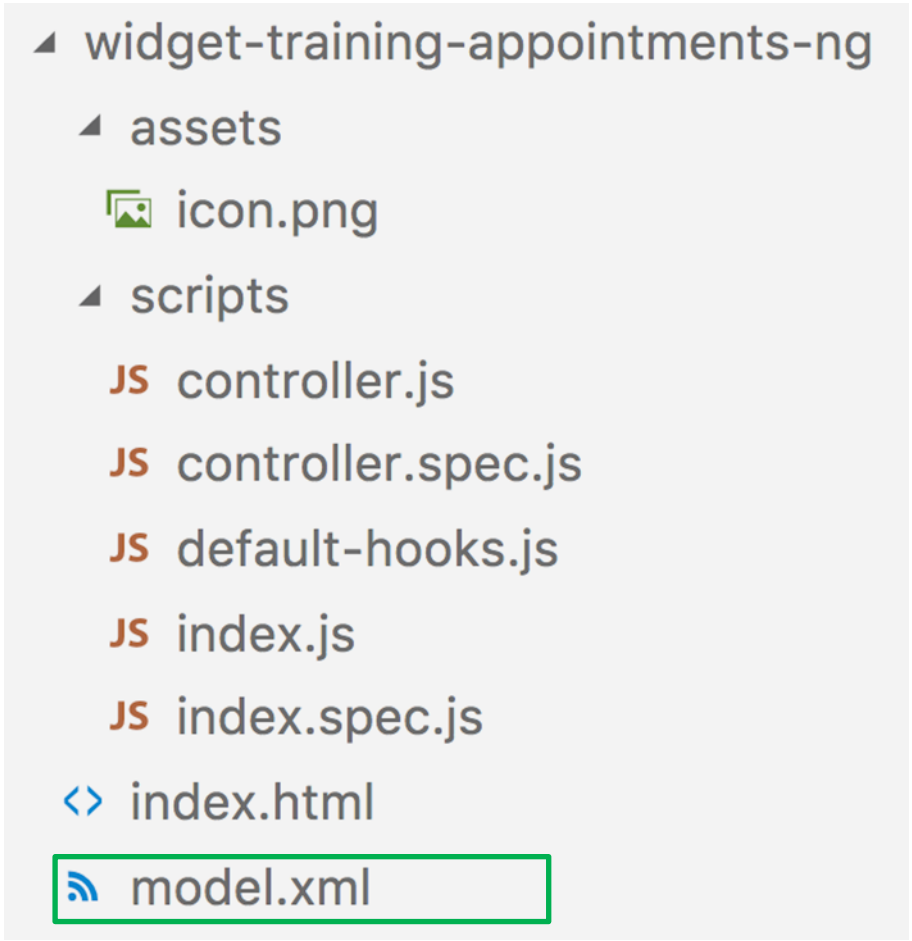
Scaffolded Widget Core



Default Hooks

Create and Expose Hooks

Scaffolded Widget Core



Model XML

Widget


ADDING WIDGET
PREFERENCES

BACKBASE

Widget Model

Widget Development

Scaffolded Widget Model

Factory

Normalize HTTP Layer

Index

Register as a Module

Inject Dependencies

Create a Custom Widget

*Generate a custom widget using bb-cli. As mentioned, the exercise is to create an **appointments** widget.*

1. Generate a Custom Widget.

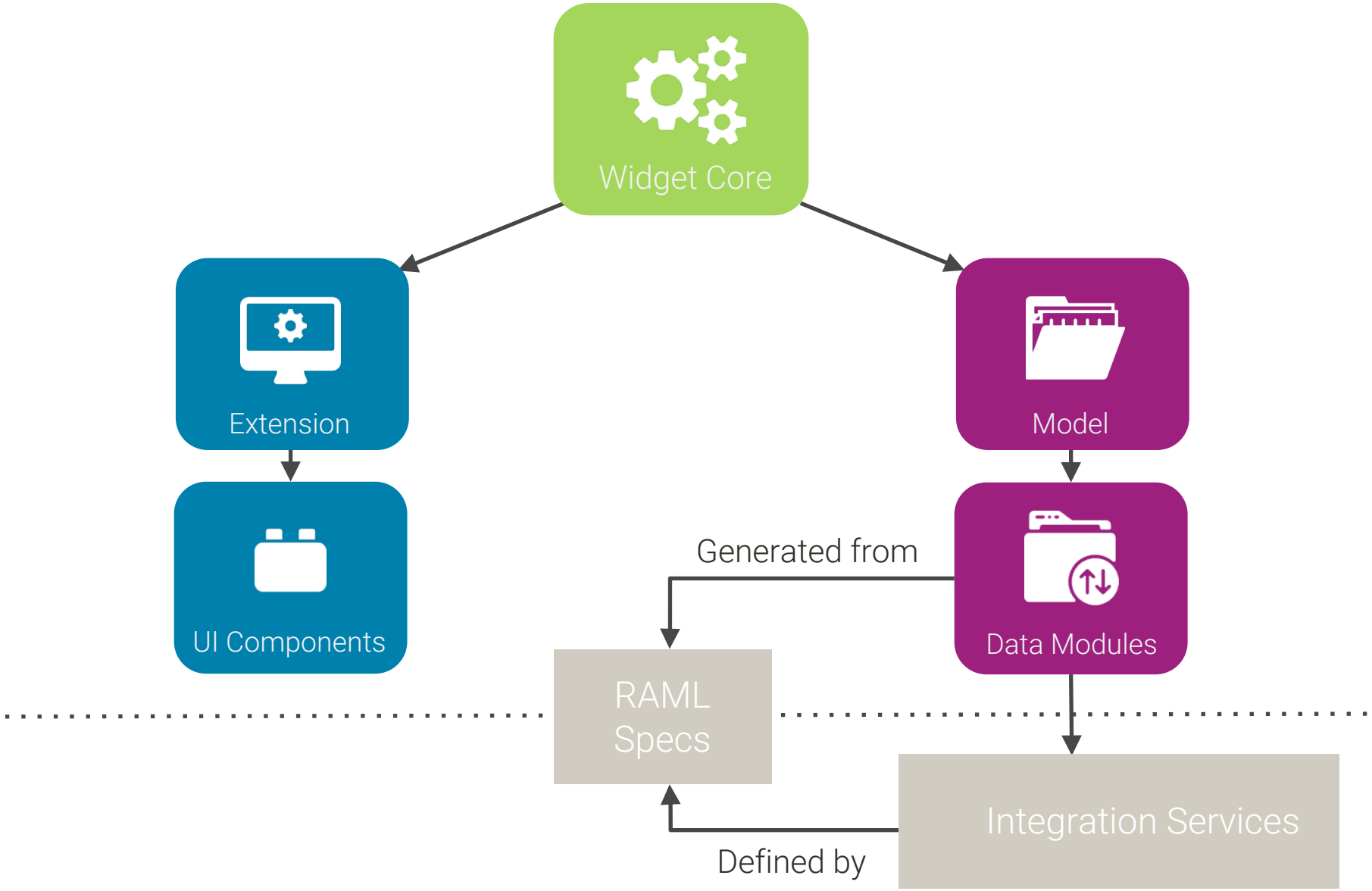
2. Put the Custom Widget on a newly created Page.

BACKBASE

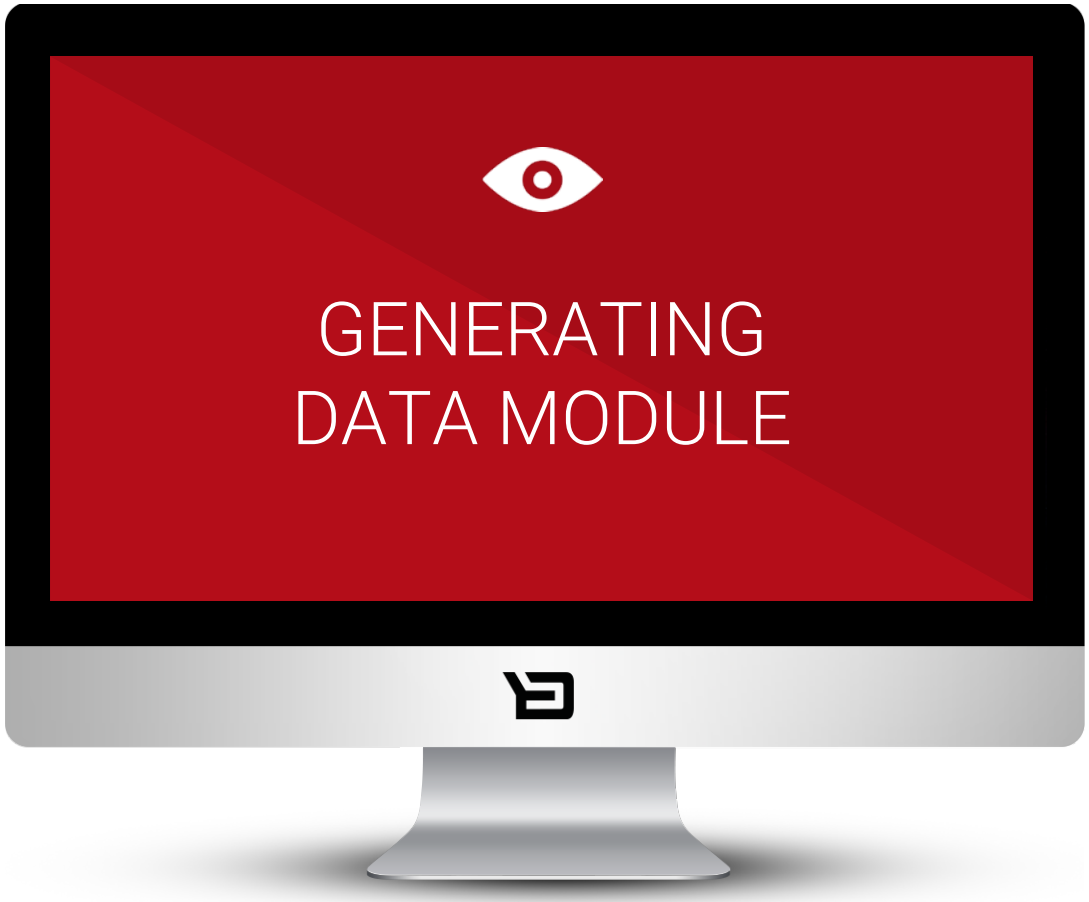
Data Modules

Widget Development

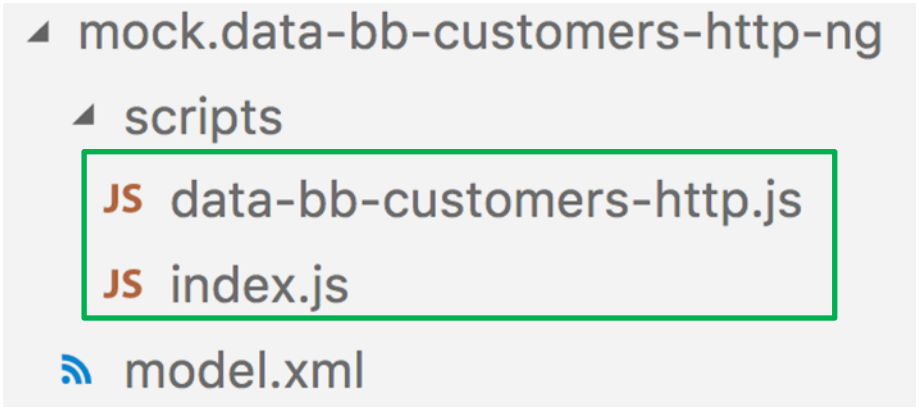
Widget Architectural Overview



- **RAML** RESTful API Modeling Language
- **bb-convert** generates a Data Module based on the specifications defined in the RAML file.
- **mock-ng** is a template that creates a mock module in mocks.


GENERATING
DATA MODULE

Generated Data Module



Provider

Communicate with REST API

Index

Register as a Module

Inject Dependencies

Configure API Routes

Generating Data Modules

Generate Data Modules based on provided RAML specifications. Use the generated Data Modules in the created Widget in the previous exercise

1. Generate Data Modules from RAML specifications
2. Use the Data Modules to serve as Mocked Data
3. Render the Mocked Data on the View

Add Sorting Functionality to the Widget

Implement the Sorting functionality to let the users sort Appointments by date in ascending / descending order

1. Add logic in the Controller to implement sorting functionality

Communicating between Widgets

Create another Widget to display selected appointment in a Modal Dialog.

1. Generate a new Widget
2. Implement the Modal Dialog View for the Widget
3. Publish the Select event from Appointments List Widget
4. Subscribe to that event from Appointment Details Widget

BACKBASE

Further Reading

- Understanding the latest **Widget Blueprint**
- Testing your code using **bb-test**
- Translate and Localize Widgets (**i18n**)

- Including **Custom Vendor Libraries**
- Configuring Base URI of Data Modules using **config-bb-provider-ng**
- **Error Handling** Guidelines

BACKBASE

Thank you!

academy@backbase.com

www.backbase.com